VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Testovanie XSLT procesorov

XSLT Processors Testing

2010 Peter Habčák

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 7. mája 2010 .....

Chcem poďakovať môjmu vedúcemu Petrovi Chovancovi za užitočné rady a čas, ktorý mi venoval. Zároveň chcem poďakovať Katedre informatiky za poskytnutie XML súborov, použitých na testovanie.

Abstrakt

Cieľom tejto práce je porovnanie výkonnosti rôznych implementácií XSLT procesorov. Prvá časť práce sa zaoberá významom jazyka XSLT, teoretickým princípom činnosti, navigáciou v XML dokumente pomocou jazyka XPath a funkciou XSLT procesora. V ďalšej časti nasleduje popis práce s rozhraniami pre prácu s XML dokumentami (a teda aj XSLT transformáciami) v jednotlivých procesoroch, ktorých zvládnutie je potrebné pre vytvorenie testovacieho rozhrania pre procesory. Následne je popísané testovanie s ohľadom na rôzne testované XML dokumenty a transformačné šablóny spolu so zhodnotením výsledkov. Záver práce sa venuje problému pamäťovej náročnosti spracovania veľmi veľkých XML súborov.

Kľúčové slová: JAXP, testovanie, transformácia, XML, XPath, XSLT procesor

Abstract

The purpose of this study is to compare different implementations of the XSLT processors. The first part deals with the importance of the XSLT language, theoretical principles, navigating in a XML document using the XPath language and functions of the XSLT processor. Following section describes how to work with the interfaces used for working with the XML documents (and therefore XSLT transformations) in the individual processors, whose mastery is required to create a test interface for processors. Subsequently there is a description of testing with regard to various tested XML documents and transformation templates, together with completion of the results. The conclusion deals with the problem of memory consumption of processing very large XML files.

Keywords: JAXP, testing, transformation, XML, XPath, XSLT processor

Zoznam použitých skratiek a symbolov

API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
GCC	–	GNU Compiler Collection
GB	–	gigabyte
GNOME	–	GNU Network Object Model Environment
GNU	–	GNU's Not Unix
HTML	–	HyperText Markup Language
JDK	–	Java Development Kit
JAXP	–	Java API for XML Processing
LLVM	–	Low Level Virtual Machine
MB	–	megabyte
PDF	–	Portable Document Format
SAX	–	Simple API for XML
SDK	–	Software Development Kit
STX	–	Streaming Transformations for XML
SVG	–	Scalable Vector Graphics
TrAX	–	Transformation API for XML
URI	–	Uniform Resource Identifier
W3C	–	World Wide Web Consortium
XHTML	–	Extensible HyperText Markup Language
XML	–	Extensible Markup Language
XMPP	–	Extensible Messaging and Presence Protocol
XQuery	–	XML Query Language
XPath	–	XML Path Language
XSD	–	XML Schema Definition
XSL-FO	–	Extensible Stylesheet Language – Formatting Objects
XSLT	–	Extensible Stylesheet Language Transformations
kB	–	kilobyte

Obsah

1 Úvod

Jazyk XML je metajazyk, ktorý nemá pevne dané tagy, ale umožňuje vývojárom vytvoriť si vlastnú definíciu typu dokumentu a teda vlastné tagy, čím vznikne nový značkovací jazyk. Táto vlastnosť ho predurčuje na široké využitie v rôznych oblastiach. V dnešnej dobe sa používa napríklad ako formát na ukladanie dokumentov v aplikáciách (asi najznámejšie je jeho použitie v kancelárskom balíku Microsoft Office od verzie 2007), v internetových protokoloch – napríklad XMPP/Jabber, slúžiaci na komunikáciu v reálnom čase, v počíta-čovej grafike – vektorový formát SVG, pri zápise matematických vzorcov – jazyk MathML. Aj jazyk XHTML je založený na XML.

Niekedy vzniká potreba z XML dokumentu vytvoriť iný dokument – HTML (potreba zobrazovať dáta z XML dokumentu ako internetovú stránku), XML s inou štruktúrou, čistý text (práca v prostredí, ktoré nie je pripravené na XML). Práve tieto činnosti má umožňovať jazyk XSLT.

Jazyk XSLT je vysokoúrovňový, preto je výkonnosť jeho implementácie závislá na konkrétnom XSLT procesore. Výkonnosť týchto implementácií je rôzna. Dôvodom sú rôzne optimalizácie, použité počas vývoja, ako aj rozdielna rýchlosť použitých programovacích jazykov. Preto pri spracovávaní veľkých XML súborov alebo veľkého množstva malých súborov môžu byť značné rozdiely v čase, potrebnom na spracovanie súboru.

Cieľom tejto práce je porovnanie rýchlosti rôznych XSLT procesorov pri spracovávaní súborov rôznej veľkosti. Testovacia zostava obsahuje ako súbory o veľkosti okolo 100 MB, tak aj malé súbory s veľkosťou iba niekoľko kB, ale vo veľkom počte.

V prvej časti práce je objasnená história jazyka a jeho význam, vysvetlený princíp fungovania transformácie a navigácia v dokumente, na ktorú sa využíva jazyk XPath. Zároveň je uvedený postup, ktorý používa XSLT procesor pri spracovávaní súboru od načítania vstupu až po zápis výstupu, a popis jednotlivých procesorov.

Ďalšia časť je venovaná výberu programovacieho jazyka, použitého na implementáciu testovacieho rozhrania a popisu rozhraní, ktoré sa používajú pri použití rôznych procesorov. Dôležitú úlohu tu zohráva rozhranie JAXP, ktoré poskytuje API pre prácu s XML v jazyku Java a COM, ktoré využíva MSXML.

Tretia časť sa venuje samotnému testovaniu procesorov. Začína popisom testovaných XML dokumentov a transformácií, ktoré sú nad nimi vykonávané. Nasleduje detailné vyhodnotenie priebehu testovania spolu s tabuľkami, v ktorých sú uvedené výsledky testovania pre jednotlivé transformácie (čas potrebný na vykonanie transformácie a rýchlosť).

V závere je uvedené stručné zhodnotenie výkonnosti jednotlivých procesorov v rámci celého testu. Časť je venovaná aj problematike pamäťovej náročnosti spracovávania veľmi veľkých XML dokumentov.

2 Jazyk XSLT

2.1 História a význam jazyka XSLT

Jazyk XSLT je projektom konzorcia W3C. Pôvodne bol jazyk vyvíjaný ako súčasť jazyka XSL, ktorý mal slúžiť na formátovanie XML dokumentov (podobne ako jazyk CSS pri HTML dokumentoch). Neskôr sa ukázalo ako žiaduce, aby sa tento jazyk rozdelil na časť vykonávajúcu transformácie (dnešné XSLT) a časť vykonávajúcu formátovanie dokumentov (dnes známa ako jazyk XSL-FO). [1]

Po niekoľkých pracovných verziách[1], vydaných v priebehu rokov 1998 a 1999 vyšla verzia 1.0 ako odporúčanie[2] 16. novembra 1999. Hlavným editorom bol James Clark. [2] Po vydaní finálnej verzie 1.0 vyšli dve pracovné verzie novej verzie 1.1 v priebehu rokov 2000 a 2001. Táto verzia však nikdy nebola uvoľnená ako finálna a namiesto toho sa vývojári sústredili na práce na verzii 2.0. [3]

Revidovaná verzia 2.0 vyšla po niekoľkých rokoch vývoja 20. januára 2007. Rolu hlavného editora prebral Michael Kay z firmy Saxonica, ktorý pracuje aj na vývoji procesora Saxon. Táto verzia priniesla rôzne vylepšenia a opravy chýb verzie 1.0. Medzi najdôležitejšie patria: [4]

- umožnenie zoskupovania elementov podľa obsahu elementu alebo atribútu, ale aj podľa vypočítanej hodnoty (napríklad dĺžka reťazca)
- výstup do viacerých dokumentov
- všetky výrazy v XPath 2.0 vracajú zoznam, čím sa mení činnosť niektorých funkcií, ktoré vo verzii 1.0 vracali v prípade viacerých súhlasiacich hodnôt vždy iba prvú
- používateľ si môže definovať vlastné funkcie, ktoré môže volať pri spracovávaní súboru
- vyhľadávanie, nahradzovanie a rozdelenie reťazca je možné realizovať pomocou regulárneho výrazu
- s premennou je možné pracovať ako s externým dokumentom (je možné v nej používať XPath výrazy)

Verzia 2.0 však má podporu iba v malom počte procesorov, viď tabuľku 1. Preto sú aj v rámci tejto práce použité jazyky XSLT a XPath vo verzii 1.0.

Vývoj jazyka XSLT je úzko spojený s jazykom XPath, ktorý sa využíva na navigáciu v rámci XML dokumentu a je podrobnejšie opísaný neskôr v rámci tejto kapitoly. Pracovné i finálne verzie oboch špecifikácií vychádzali vždy v rovnaký deň.

Jazyk XSLT má široké možnosti využitia pri práci s XML dokumentami. Medzi jeho najbežnejšie využitie patrí: [5]

[1] označených ako *Working Draft*

[2] angl. *Recommendation* – označenie pre finálnu špecifikáciu, vydanú W3C

[3] externé knižnice pridávajú podporu rôznych jazykov, napr. C++, PHP, Python ...

	Podporované jazyky	XSLT 1.0	XSLT 2.0	XPath 1.0	XPath 2.0
Xalan	Java, C++	Áno	Nie	Áno	Nie
Saxon	Java, C	Nie	Áno	Nie	Áno
Xt	Java	Áno	Nie	Áno	Nie
MSXML	C++, .NET	Áno	Nie	Áno	Nie
Sablotron	C++, PHP	Áno	Nie	Áno	Nie
libxml	C[3]	Áno	Nie	Áno	Nie

Tabuľka 1: Porovnanie vlastností testovaných procesorov

- Transformácia XML dokumentu na XHTML alebo HTML dokument a jeho následné zobrazenie vo webovom prehliadači. Zdrojové kódy internetovej stránky teda môžu byť zapísané v jazyku XML, ktorý má inú štruktúru ako HTML (napr. jednoduchšiu na pochopenie pre laika) a tento dokument bude transformovaný na HTML buď prostredníctvom XSLT procesora na webovom serveri, alebo priamo prostredníctvom internetového prehliadača u klienta (XSLT transformáciu podporujú všetky novšie verzie prehliadačov – Mozilla Firefox od verzie 3, Internet Explorer od verzie 6, Opera od verzie 9, Safari od verzie 3 a Google Chrome od verzie 1 [6]).
- Extrahovanie textu z XML dokumentu a jeho následné použitie v prostredí, ktoré nepracuje s XML. Ďalej je možné tento text priamo vložiť do nejakého textového formátu a tak vytvoriť napríklad príkazy jazyka SQL, ktoré vložia údaje z XML dokumentu do databázy.
- Prevedenie XML dokumentu z jedného formátu do iného (napríklad OpenDocument na SVG). Tu patrí aj prevod do formátu XSL–FO, ktorý je možné pomocou procesora XSL–FO previesť na dokument, pripravený na tlač (napríklad PDF).
- Výber iba niektorých dát z XML dokumentu (užitočné napríklad pri mnohojazyčnom dokumente, z ktorého chceme vybrať iba jeden jazyk).

2.2 Fungovanie jazyka a navigácia v dokumente

Jazyk XSLT je založený na princípe šablóny, ktorá obsahuje vzor[4], ktorý obsahuje názov elementu v rámci XML dokumentu, a telo, ktoré definuje, ako sa bude spracovávať element dokumentu, ktorý zadanému vzoru vyhovuje. [2] Tieto šablóny môžu byť do seba zanorené. Ukážka jednoduchej transformácie:

[4] angl. *pattern*

```
<?xml version="1.0" encoding="UTF-8"?>
<stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<output method="text"/>

  <xsl:template match="/">Knihy
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="kniha">
    ---<xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="nazov">
    Kniha:<xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="autor">
    Autor:<xsl:apply-templates/>
  </xsl:template>

</stylesheet>
```

Výpis 1: Ukážka transformácie v jazyku XSLT

Na začiatku XSLT súboru je definícia použitého menného priestoru[5], ktorý slúži na definovanie tagov jazyka XSLT. Štandardne začínajú všetky tagy prefixom xsl: (môže však byť použité ľubovoľné meno). Pri spracovávaní sa v stromovej štruktúre vyhľadá šablóna pre koreň dokumentu. Naň sa potom aplikuje telo šablóny.

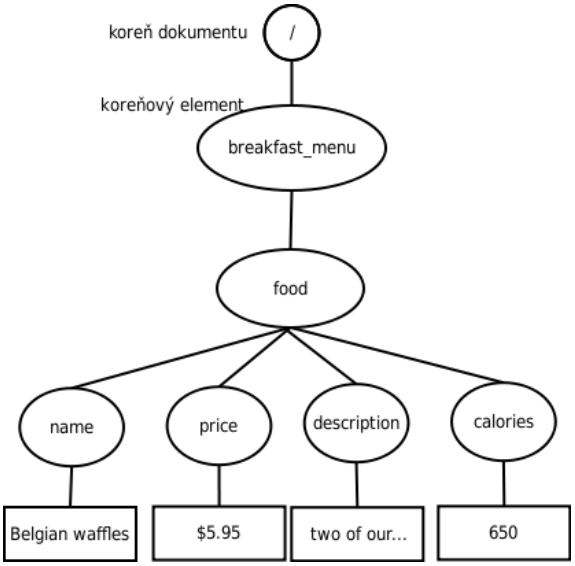
Volaním tagu <xsl:apply-templates/> sa vyhľadávajú šablóny použiteľné pre všetkých priamych potomkov tohto uzlu (podľa hodnoty atribútu *match*, ktorý obsahuje výraz jazyka XPath). Toto správanie sa dá ovplyvniť pomocou atribútu *select*, ktorý umožňuje vybrať konkrétneho potomka. [2]

Pri navigácii v dokumente sa využíva dotazovací jazyk XPath. Tento jazyk bol vyvinutý na adresovanie častí XML dokumentu. Ďalším využitím je testovanie, či zadanému vzoru zodpovedá nejaký uzol XML dokumentu. Práve táto vlastnosť sa využíva v jazyku XSLT. [7]

Jazyk XPath vníma XML dokument ako strom uzlov. Príklad takéhoto stromu je na obrázku 1. Koreňom stromu nie je žiaden element v dokumente. Je to umelý uzol, ktorého priamym potomkom je koreňový (hlavný) element dokumentu (v zobrazenom príklade je to element *breakfast_menu*). [7]

Ďalšími dôležitými uzlami stromu sú uzly, reprezentujúce jednotlivé elementy dokumentu. Ku každému z týchto uzlov môže byť pripojená množina uzlov, ktoré reprezentujú atribúty a menné priestory elementu. Tento uzol je rodič pre každý z nich, avšak žiaden z nich nie je jeho priamy potomok. Priamym potomkom môže byť iný element, textový uzol, uzol pre komentár alebo uzol s inštrukciami pre spracovanie. [7]

[5] angl. *namespace*

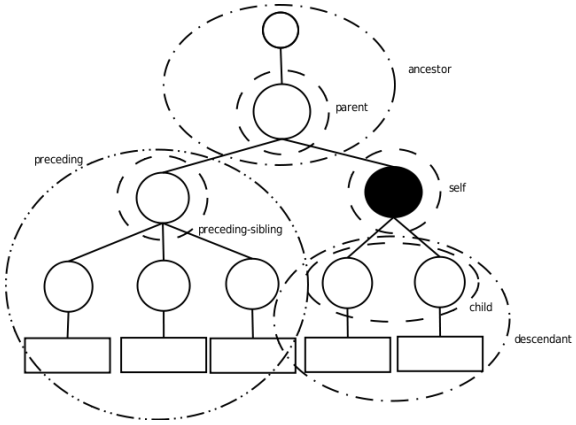


Obr. 1: XML dokument ako strom

Textový uzol je obsah uzlu, ktorý obsahuje textové dáta (na obrázku je vyznačený ako obdĺžnik).

Dôležité pre navigáciu v dokumente sú osi, ktoré určujú vzťah medzi uzlami v rámci dokumentu. V jazyku XPath sú definované nasledujúce osi: [7]

- *self* – obsahuje odkaz na uzol sám
- *child* – obsahuje priamych potomkov uzla
- *descendant* – obsahuje potomkov uzla (priamych i nepriamych potomkov), nikdy neobsahuje uzol atribútu alebo menného priestoru
- *parent* – obsahuje rodiča uzla (každý uzol má iba jedného rodiča)
- *ancestor* – obsahuje predkov uzla (rodiča, rodiča rodiča a tak ďalej), vždy obsahuje aj koreňový uzol
- *preceding-sibling* – obsahuje predchádzajúcich súrodencov uzla (uzly, ktoré majú rovnakého rodiča a už boli spracované)
- *following-sibling* – obsahuje nasledujúcich súrodencov uzla (uzly, ktoré majú rovnakého rodiča a ešte budú spracované)
- *preceding* – obsahuje všetky predchádzajúce uzly dokumentu (uzly, ktoré boli spracované), okrem predkov a uzlov s atribútmi a mennými priestormi
- *following* – obsahuje všetky nasledujúce uzly dokumentu (uzly, ktoré ešte budú spracované), okrem potomokov a uzlov s atribútmi a mennými priestormi



Obr. 2: Osi v jazyku XPath 1

- *ancestor-or-self* – obsahuje predkov uzla a uzol sám
- *descendant-or-self* – obsahuje potomkov uzla a uzol sám

Osi sa často zapisujú pomocou skráteného zápisu (často používané príklady): [5]

Skrátený zápis	Neskrátený zápis
meno	child::meno
../meno	parent::meno
//meno	descendant::meno
@meno	attribute::meno
*	child::*

Tabuľka 2: Skrátený a neskrátený zápis XPath osí

Štandardne sa v XSLT dokumentoch používa os *child*. [7]

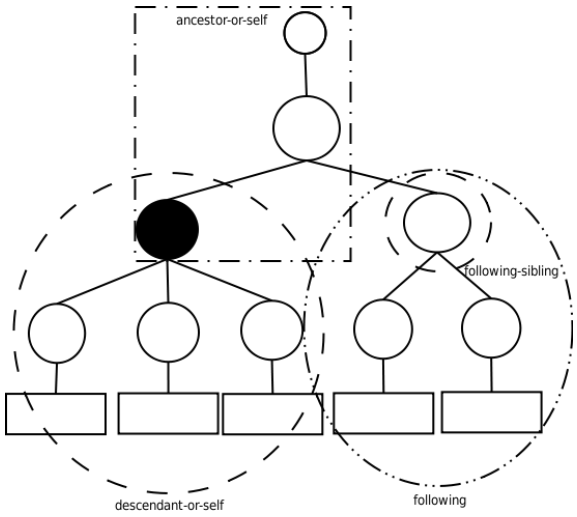
Graficky sú osi znázornené na obrázkoch 2 a 3. Čiernou výplňou je zvýraznený element, ktorý sa práve spracováva.

2.3 Spracovanie dokumentu pomocou XSLT procesora

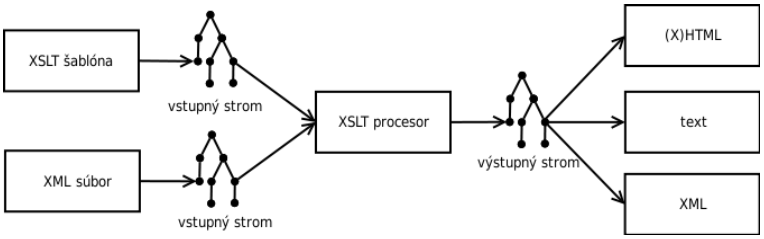
Na obrázku 4 je zobrazený princíp transformácie. Tento proces začína načítaním vstupného XML súboru. Tento dokument nemusí byť validny, je však nevyhnutné, aby bol dobre štrukturovaný [6]. To kladie na dokument tieto požiadavky: [8]

- Dokument obsahuje aspoň jeden element.
- Dokument obsahuje koreňový element, v ktorom sú zanorené ostatné elementy.

[6] angl. *well formed*



Obr. 3: Osi v jazyku XPath 2



Obr. 4: Princíp XSLT transformácie

- Každý element musí začínať otváracím tagom a končiť koncovým tagom. Prázdny element (neobsahuje iný element ani textové dáta) musí byť ukončený znakom /.
- Atribúty elementu musia byť zapísané v úvodzovkách.
- Elementy musia byť do seba správne zanorené.
- Rozlišujú sa veľké a malé písmená.

Príklad dokumentu, ktorý nie je dobre štrukturovaný (chýba koncový tag a atribút nie je v úvodzovkách):

```
<?xml version="1.0" encoding="UTF-8"?>
<knihy>
 <kniha>
   <autor>J. R. R. Tolkien
   <cena mena=EUR>5.95</cena>
   <nazov>Pan prstenov: Dve veze</nazov>
 </kniha>
</knihy>
```

Výpis 2: Tento XML súbor nie je well-formed

Ďalším súborom, ktorý je potrebný na vykonanie transformácie, je šablóna s transformáciou. Táto musí byť zapísaná v jazyku XSL.

Po načítaní je dokument prevedený na stromovú štruktúru pomocou XML parsera. Pritom môže vykonať rôzne optimalizácie, ktoré ovplyvňujú výslednú rýchlosť práce s týmto stromom a jeho spotrebu pamäte. [9]

Potom takto vytvoreným stromom prechádza XSLT procesor pomocou algoritmu prehľadávania do hĺbky a pre každý element vyberá vhodnú šablónu z transformačného súboru. Potom zapisuje elementy do výstupného stromu podľa pravidiel, zapísaných v šablóne. Platí tu pravidlo, že vstupný a výstupný strom sú oddelené a procesor nikdy nemodifikuje vstupný strom. [2]

Po vytvorení výstupného stromu procesor pomocou procesu serializácie zapíše tento strom ako XML dokument na požadovaný výstup (na obrazovku, do súboru).

Okrem spracovania pomocou samostatného XSLT procesora je možné ho použiť aj priamo vo webovom prehliadači, pridaním nasledujúceho riadku do XML súboru:

<?xml−stylesheet href="stylesheet.xsl"type="text/xsl"?>

Vtedy je XML dokument spracovaný XSLT procesorom, ktorý je zabudovaný do prehliadača a jeho výstup je zobrazený ako výsledná stránka.

2.4 XSLT procesory

2.4.1 Xalan-J

Tento procesor bol pôvodne vyvíjaný spoločnosťou IBM pod názvom LotusXSL. [10] Neskôr bol uvoľnený ako projekt s otvoreným zdrojovým kódom, vyvíjaný v rámci Apache Software Foundation. Implementuje jazyky XSLT verzie 1.0, XPath verzie 1.0 a rozhranie JAXP verzie 1.3. Podporovaný programovací jazyk je Java. Je vydaný pod licenciou Apache License 2.0 a dodávaný spolu s XML parserom Xerces-J, ktorý podporuje

rozhrania DOM Level 3 a SAX Level 2. Môže byť však spojený s akýmkoľvek XML parserom, ktorý podporuje rozhranie JAXP 1.3. [11] Okrem tohto rozhrania je procesor poskytovaný aj ako samostatná aplikácia vo formáte jar, pomocou ktorej je možné vykonávať transformácie z príkazového riadka.

2.4.2 Xalan-C++

Xalan-C++ je verziou procesora Xalan, ktorá je určená je programovací jazyk C++. Pôvodne bol vyvíjaný spoločnosťou IBM pod názvom LotusXSL. [10] Neskôr bol uvoľnený ako projekt s otvoreným zdrojovým kódom. Posledná uvoľnená verzia je 1.10 z roku 2004. Podporuje jazyky XSLT 1.0 a XPath 1.0. [12] Je dodávaný spolu s XML parserom Xerces-C++. Poskytovaný je aj ako samostatná aplikácia pre príkazový riadok.

2.4.3 Saxon

Procesor Saxon vyvíja Michael Kay z firmy Saxonica Limited, ktorý pracoval ako editor na špecifikácii jazyka XSLT 2.0. Pôvodne (až do verzie 6.5.5) procesor podporoval jazyk XSLT 1.0 a bol dostupný iba pre jazyk Java. Táto vývojová vetva je dnes v stave udržovania bez pridávania nových funkcií. V aktuálnej vývojovej vetve je podporovaný jazyk Java a platforma .NET. Verzia pre Javu podporuje rozhranie JAXP.

V roku 2004 došlo k rozdeleniu na dve verzie: Saxon-B (*Basic*) s otvoreným zdrojovým kódom a Saxon-SA (*Schema-Aware*) ako komerčný produkt. Prednosťou verzie SA bola možnosť validovať dokument oproti XSD. [13]

Od verzie 9.2 sú dostupné tri verzie: [14]

- *Home Edition* – je produkt s otvoreným zdrojovým kódom, dostupný pod licenciou Mozilla Public License. Je to priamy nástupca verzie Saxon-B. Podporuje XSLT 2.0, XPath 2.0 a XQuery 1.0 na základnej úrovni, ktorú vyžaduje W3C.
- *Professional Edition* – komerčný produkt, k funkciám verzie Home Edition pridáva podporu pre rozšírenia, podporu jazyka XQuery 1.1 a podporu pre externé objektové modely.
- *Enterprise Edition* – komerčný produkt, nástupca verzie Saxon-SA. Podporuje XSD 1.0, optimalizátor dotazov, kompiláciu kódu XQuery do bytekódu Javy a čiastočnú podporu pre pracovné verzie[7] XSD 1.1, XSLT 2.1 a XQuery 1.1.

Je poskytovaný aj ako jar archív pre vykonávanie transformácií z príkazového riadka. V minulosti existovala aj verzia *Instant Saxon*, ktorá bola určená pre jednoduchú inštaláciu v prostredí Microsoft Java.

2.4.4 Xt

Procesor Xt bol jeden z prvých XSLT procesorov. Vznikol v čase, keď bol jazyk XSLT ešte v štádiu pracovnej verzie (prvá alfa verzia vyšla v auguste 1998). [1] Pôvodne bol

[7] *Draft* verzie

vyvíjaný Jamesom Clarkom, ktorý jeho vývoj ukončil verziou verziou 19991105, ktorá bola založená na ešte nehotovej verzii XSLT PR-xslt-19991008. Neskôr sa vývoja ujal Bill Lindsey. Pridal podporu pre finálnu verziu XSLT 1.0 a v ďalších verziách aj pre rozhranie JAXP. Procesor má podporu pre rozširujúce funkcie. Dodávaný je spolu s XML parserom XP. [15] Poskytovaný je aj jar archív pre vykonávanie transformácií z príkazového riadka.

2.4.5 MSXML

Knižnicu MSXML vyvíja spoločnosť Microsoft. Je to komplexný balík pre prácu s XML, obsahuje XML parser, XSLT procesor (podporuje XSLT 1.0 a XPath 1.0) a podporu pre XML schema. [16] Prvé dve verzie (1.0 a 2.0) boli distribuované ako súčasť prehliadača Internet Explorer. Tieto verzie, spolu v verziami 2.5 a 2.6, sú označené ako zastaralé a oficiálne nie sú podporované. [17]

Verzia 3.0 bola dodávaná spolu s operačným systémom Windows XP. Táto verzia je používaná prehliadačom Internet Explorer od verzie 6.0 pri parsovaní XML dokumentov. Verzia 4.0 bola vydaná ako samostatné SDK, ktoré nenahradzuje staršiu verziu. Verzia 5.0 bola vyvinutá výhradne pre použitie v balíku Microsoft Office (dodávaná s balíkmi Office 2003 a 2007). Nebola k nej vydaná žiadna dokumentácia. [17]

Najnovšia verzia je 6.0, ktorá je súčasťou operačného systému Windows Vista a Windows Seven. Túto verziu je možné používať paralelne so staršími verziami, nakoľko ich plne nenahradzuje.

Okrem SDK je osobitne dostupná aj aplikácia, ktorá umožní spracovanie dokumentov z príkazového riadka.

2.4.6 Sablotron

Sablotron je projekt s otvoreným zdrojovým kódom, dostupný pod dvojitou licenciou (Mozilla Public License alebo GNU General Public License). Podporuje XSLT 1.0, XPath 1.0 a DOM Level2. Bol vytvorený firmou Ginger Alliance. Sablotron bol využívaný ako štandardný XSLT procesor v jazyku PHP4. Ako XML parser využíva Expat. Dostupná je aj aplikácia, ktorá umožňuje spracovanie dokumentov z príkazového riadka.

2.4.7 libxslt

libxslt je projekt s otvoreným zdrojovým kódom, vydaný pod licenciou MIT. Ako parser používa knižnicu libxml. Pôvodne ju napísal Daniel Veillard v jazyku C pre projekt GNOME, nie je však závislá na žiadnych knižniciach z toho projektu.[18] Táto knižnica je súčasťou projektu WebKit a tak je využitá na XSLT transformácie v prehliadačoch Safari a Google Chrome. [19] Okrem API je dostupná aj aplikácia *xsltproc*, ktorá umožňuje vykonávať transformácie z príkazového riadka.

Postupom času vznikli knižnice, ktoré umožnili použiť ju i v rámci širokej škály programovacích jazykov (napr. C++, C#, PHP, Python, Ruby). Pre účely testovania je použitý projekt *xmlwrapp*, ktorý umožňuje používať libxslt v jazyku C++.

2.4.8 Unicorn XLST processor

Tento procesor vyvíja firma Unicorn Enterprises, ktorá sa špecializuje na správu dokumentov, intranetové a XML technológie (vyvíja napríklad aj SVG a XML-FO procesor). Tento procesor je dodávaný v dvoch verziách: *Standard* a *Database*. Verzia Database umožňuje prístup k dátam uloženým v relačných databázach. Procesor bol vyvinutý v rokoch 1999–2001, neskôr bol vývoj opustený, pretože firma sa zamerala na vlastný jazyk *Unicorn Extensible Stylesheets*.**??**

Procesor je napísaný v jazyku C++ a dostupný ako aplikácia pre príkazový riadok, neobsahuje API. Posledná verzia je 1.50.00.

3 Implementácia testovacieho rozhrania

3.1 Výber jazyka pre testovacie rozhranie

Pre implementáciu testovacieho rozhrania som vybral programovacie jazyky Java a C++. Vybral som obidva jazyky preto, že procesory, ktoré som testoval, sú väčšinou dostupné iba pre jeden z týchto jazykov (výnimkou je Xalan, ktorý je dostupný pre Javu aj C++).

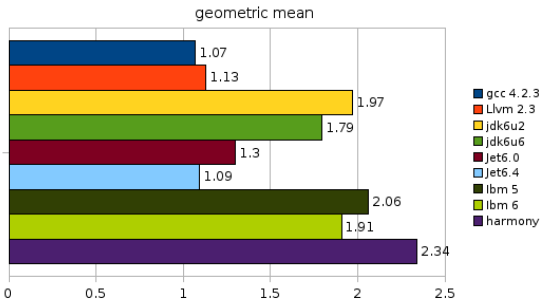
Java je objektovo orientovaný programovací jazyk, ktorý bol vyvinutý firmou Sun Microsystems. Jeho hlavnou výhodou je, že program skompilovaný do bytekódu je možné spustiť na každom počítači, kde beží virtuálny stroj Javy bez potreby úpravy kódu alebo rekompilácie.

C++ je objektovo orientovaný programovací jazyk, ktorý vznikol v 80. rokoch 20. storočia v Bellovych laboratóriách ako rozšírenie jazyka C s cieľom pridať do tohto jazyka prvky objektového programovania. [21]

Porovnanie rýchlosti jazykov Java a C++ (prípadne C) a teda aj vplyv rýchlosti jazyka na vykonávané činnosti je náročné, pretože rýchlosť behu závisí od konkrétnej implementácie programovacieho jazyka. Existujú rozdiely v rýchlosti programov skompilovaných rôznymi kompilátormi C/C++ a takisto sú rozdiely v rýchlosti pre programy bežiace pod rôznymi virtuálnymi strojmi pre jazyk Java. Programy v Jave môžu byť v niektorých prípadoch rýchlejšie vďaka použitiu Just-In-Time kompilácie, kde sa časti bytekódu kompilujú do natívneho strojového kódu počas behu programu, preto môžu využiť optimalizáciu pre procesor, na ktorom bežia. [22] Pri tomto testovacom rozhraní však Just-In-Time nie je veľkou výhodou, pretože rozhranie pre jazyk C++ je kompilované na rovnakom počítači, ako bude spustené, preto má kompilátor k dispozícii všetky optimalizácie.

Na internete je dostupných mnoho porovnaní rýchlosti jazykov Java a C/C++. Ja som si pre účely znázornenia vybral test od Stefana Krausa [23], ktorý porovnáva rýchlosť programov skompilovaných týmito prostrediami:

- GCC – kompilátor pre C/C++ s otvoreným zdrojovým kódom, používaný hlavne v prostredí Linux/Unix
- LLVM – virtuálny stroj pre jazyk C/C++, ako kompilátor používa GCC a umožňuje využívať Just-In-Time
- Sun JDK – originálna implementácia jazyka Java od firmy Sun s použitím Hotspot server
- IBM JDK – implementácia jazyka Java od firmy IBM, nie je dostupná pre operačný systém Windows
- Excelsior JET – komerčná implementácia jazyka Java s dobrou podporou pre optimalizácie
- Apache Harmony – implementácia jazyka Java od Apache Software Foundation so slobodnou licenciou



Obr. 5: Výsledky porovnania rýchlosti Javy a C, zdroj: [23]

Tento test obsahoval viacero čiastkových testov, ktoré pochádzajú z projektu *The Computer Language Benchmarks Game*. Výsledky testu sú zobrazené na obrázku 5, nižšia hodnota je lepšia. Ako je možné vidieť na obrázku, program kompilovaný pomocou GCC dosiahol lepšie výsledky ako program spustený v Sun JDK, ktoré som použil v testovacom rozhraní. Preto je možné predpokladať, že aj toto bude mať vplyv na rýchlosť procesorov.

3.2 Práca s XML v jazyku Java – rozhranie JAXP

Všetky testované procesory pre jazyk Java podporujú rozhranie JAXP, ktoré bolo vyvinuté ako platforma, ktorá uľahčuje prácu a poskytuje rovnaké rozhranie pre prácu s XML súbormi, bez ohľadu na použité nástroje (XML parser, XSLT procesor). [24] JAXP neposkytuje žiadne nové funkcie na parsovanie súborov alebo vykonávanie transformácií. Pracuje nad iným rozhraním (SAX, DOM) a bez neho nie je schopné vykonávať žiadne operácie.

Verzia 1.0, ktorá vyšla v roku 2000, obsahovala iba rozhranie pre XML parsery. Neskôr bol do rozhrania pridaný kód z už existujúceho projektu TrAX a vďaka nemu od verzie 1.1 JAXP podporuje aj XSLT transformácie. [25]

Pri použití rozhrania JAXP programátor nepoužíva priamo triedy konkrétneho XML parsera či XSLT procesora, ale používa abstraktné triedy, ktoré definuje toto rozhranie. Tieto abstraktné triedy sú následné implementované v konkrétnom programe, ktorý má starosti parsovanie či transformáciu. [26]

Výpis 3 predstavuje jednoduchú implementáciu XSLT transformácie pomocou JAXP.

```
import java.io.*;

public class JaxpDemo {

    public static void main(String[] args)
            throws javax.xml.transform.TransformerException {

        File xmlFile = new File(args[0]);
        File xsltFile = new File(args[1]);
```

```
            javax.xml.transform.Source xmlSource =
                    new javax.xml.transform.stream.StreamSource(xmlFile);
            javax.xml.transform.Source xsltSource =
                    new javax.xml.transform.stream.StreamSource(xsltFile);
            javax.xml.transform.Result result =
                    new javax.xml.transform.stream.StreamResult(System.out);

            javax.xml.transform.TransformerFactory transFact =
                    javax.xml.transform.TransformerFactory.newInstance( );

            javax.xml.transform.Transformer trans =
                    transFact.newTransformer(xsltSource);

            trans.transform(xmlSource, result);
        }
    }
```

Výpis 3: Príklad použitia rozhrania JAXP

Triedy, ktoré zabezpečujú XSLT transformáciu, sú súčasťou balíka java.xml.transform . Rozhranie JAXP umožňuje reprezentovať XML dokument ako DOM dokument, sériu udalostí spracovaných rozhraním SAX alebo priamo ako súbor. Podobne rôznorodá reprezentácia je možná aj u spracovaného výstupu. [25]

Pre transformácie je najdôležitejšou triedou, ktorá reprezentuje priamo XSLT procesor, trieda javax.xml.transform.TransformerFactory. Jej nastavenie pre jednotlivé procesory je nasledujúce:

- Xalan-Java – org.apache.xalan.processor.TransformerFactoryImpl
- Saxon – net.sf.saxon.TransformerFactoryImpl
- Xt – com.jclark.xsl.trax.TransformerFactoryImpl

Nastavenie potrebného XML parsera a XSLT procesora sa vykonáva pomocou nastavenia *System properties*. To je možné vykonať viacerými spôsobmi: [25]

- Pri spustení výsledného programu

  java −Djavax.xml.transform.TransformerFactory=[transformer.impl.**class**] program.jar
- Zapísaním do súboru *lib/jaxp.properties*
- Zapísaním priamo do kódu programu

3.3 Práca s MSXML – rozhranie COM

V rozhraní COM je, na rozdiel od klasických objektových jazykov, rozhranie[8] nie je súčasťou objektu, ale je implementované ako samostatná trieda, ktorá obsahuje iba virtuálne

[8] *interface*

metódy. Tieto sú implementované až v rámci objektu. Podľa konvercie názvy rozhraní začínajú písmenom I. [27]

Toto rozhranie je používané aj v rámci MSXML. Na výpise 4 je zobrazený jednoduchý príklad na načítanie vstupných súborov, vykonanie transformácie a zápis do výstupného súboru.

```
#include <iostream.h>

#import <msxml4.dll>
using namespace MSXML2;

int main(int argc, char* argv[])
{

HRESULT hResult = S_OK;

hResult = CoInitialize (NULL);

try
{
 IXMLDOMDocument2Ptr spDocSource;
 IXMLDOMDocument2Ptr spDocResult;
 IXMLDOMDocument2Ptr spDocStylesheet;
 struct IDispatch * pDispatch;

 spDocSource.CreateInstance(__uuidof(DOMDocument40));
 spDocResult.CreateInstance(__uuidof(DOMDocument40));
 spDocStylesheet.CreateInstance(__uuidof(DOMDocument40));

 spDocSource->async = VARIANT_FALSE;
 spDocSource->load("HelloWorld.xml");

 spDocStylesheet->async = VARIANT_FALSE;
 spDocStylesheet->load("HelloWorld.xsl");

 spDocResult->QueryInterface(IID_IDispatch, (void **) &pDispatch);
 VARIANT vResultDoc;
 vResultDoc.vt = VT_DISPATCH;
 vResultDoc.pdispVal = pDispatch;

 spDocSource->transformNodeToObject(spDocStylesheet, vResultDoc);

 spDocResult->save("ResultDocument.xml");
}

catch (_com_error &e)
{
 cerr << "COM Error" << endl;
 cerr << "Message = " << e.ErrorMessage() << endl;
 return 1;
}

CoUninitialize () ;
```

```
    return 0;
}
```

Výpis 4: Príklad použitia rozhrania COM

Na začiatku kódu je povinná inicializácia rozhrania. Vstupné a výstupný objekty sú typu DOMDocument40 (40 označuje verziu MSXML). Pri načítavaní vstupných súborov sa vykonáva parsovanie.

vResultDoc je objekt špeciálneho dátového typu VARIANT, ktorý pochádza z jazyka VisualBasic a ktorý sa v prostredí COM využíva pre označenie „obálky“ pre premennú, ktorej dátový typ ešte nepoznáme.

Na konci, po vykonaní transformácie, nasleduje povinné odinicializovanie.

4 Experimenty

4.1 Informácie o testovaní

V mojich experimentoch som sa rozhodol testovať XSLT transformácie procesorov:

- Xalan-Java 2.7.1
- Saxon 9.2 Home Edition Java
- MSXML 4.0
- Xt 20051206
- libxslt 1.1.26

Xalan-C++ sa mi nepodarilo zaradiť do testovacieho rozhrania. Napriek tomu, že samostatná aplikácia fungovala správne, pri využití API sa vždy zobrazilo chybové oznámenie:

```
 error LNK2019: unresolved external symbol "public: static class
xercesc_2_8::MemoryManager ...
```

Na zobrazenie tohto chybového hlásenia stačilo importovať hlavičkový súbor
<xalanc/XalanTransformer/XalanTransformer.hpp>
Takisto som netestoval procesor Sablotron, ktorý nemá podporu pre jazyk C++ (iba pre jazyk C) a jeho vývoj po opustení úlohy XSLT procesora v jazyku PHP ustal.

Procesory som testoval na počítači s parametrami (iba parametre, ovplyvňujúce testovanie):

- Procesor – Intel Pentium Dual Core E5200
- Operačná pamäť – 2 GB
- Operačný systém – Microsoft Windows XP Service Pack 3

Prvý testovaný súbor obsahuje informácie z databázy SwissProt, ktorá obsahuje informácie o funkciách proteínov, ich doménovej štruktúre, variantoch. [28] Veľkosť súboru je 112,129 MB.

Štruktúra súboru (elementy, za ktorými sú tri bodky, sa opakujú):

```
<?xml version="1.0" encoding="UTF−8"?>
<root>
<Entry id="" class="" mtype="" seqlen="">
  <AC></AC>
  <Mod date="" Rel="" type=""></Mod>
  ...
  <Descr></Descr>
  <Species></Species>
  <Org></Org>
  ...
```

```
<Ref num="" pos="">
  <Comment></Comment>
   ...
  <DB></DB>
  <MedlineID></MedlineID>
  <Author></Author>
   ...
  <Cite></Cite>
</Ref>
   ...
<EMBL prim_id="" sec_id=""></EMBL>
<INTERPRO prim_id="" sec_id=""></INTERPRO>
...
<PFAM prim_id="" sec_id="" status=""></PFAM>
...
<Keyword></Keyword>
...
<Features>
  <DOMAIN from="" to="">
    <Descr></Descr>
  </DOMAIN>
   ...
  <BINDING from="" to="">
    <Descr></Descr>
  </BINDING>
   ...
</Features>
</Entry>
...
</root>
```

Výpis 5: Štruktúra súboru swissprot.xml

Na testovanie malých súborov bol zvolený projekt Wikipedia XML Corpus, ktorý obsahuje heslá z internetovej encyklopédie Wikipedia. [29] Z tohoto archívu bolo vybraných 5000 súborov, ktoré boli testované. Tieto súbory mali celkovú veľkosť 99,98 MB. Priemerná veľkosť súboru je 20,476 kB.

Tieto súbory majú menej pravidelnú štruktúru. Napríklad element *collectionlink* môže byť zanorený v mnohých ďalších elementoch.

```
<?xml version="1.0" encoding="UTF-8"?>
<article>
  <name id=""></name>

  <conversionwarning></conversionwarning>
  <body>

    <figure>
      <image xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"  xlink:href =""
            xlink:actuate =""  xlink:show="">
      </image>
      <caption>
      </caption>
```

```
</figure>
 ...

<emph3></emph3>
 ...

<emph2><emph2>
 ...

< collectionlink  xmlns:xlink=" http: // www.w3.org/1999/xlink" xlink:type="simple"  xlink:href =""">
 ...

<section>
  < title ></title >
</section>



<template name=""></template>
 ...
<languagelink lang=""></languagelink>
 ...
</section>
</body>

</ article >
```

Výpis 6: Štruktúra súboru 1000.xml

Testoval som dve transformácie. Prvou bolo vybratie textových informácií z XML dokumentu. V prípade Wikipedia XML Corpus sa jednalo o odstránenie informácií o jazykových odkazoch, vnútorných odkazoch na iné súbory a obrázky (v textovom formáte nemajú zmysel). V prípade SwissProt sa jednalo o vytvorenie textovej reprezentácie dát spolu s popiskom k jednotlivým častiam súboru.

Ďalšou transformáciou bolo podmienené spracovanie na základe hodnoty atribútu. Elementy, ktoré nesplňovali požadovanú hodnotu atribútu, neboli spracovávané vôbec.

4.2 Vybratie textových informácií z dokumentu

Ako je možné vidieť v tabuľke 3 a v prehľadnej podobe na obrázku 6, pri spracovaní veľkého súboru bol najrýchlejší procesor Saxon, ktorý tento súbor spracoval za necelých 10,42 sekúnd. Druhý najrýchlejší bol Xmlwrapp, ktorý súbor spracovával 11,98 sekúnd, Xt sa udržal v prvej trojici s časom 14,14 sekúnd.

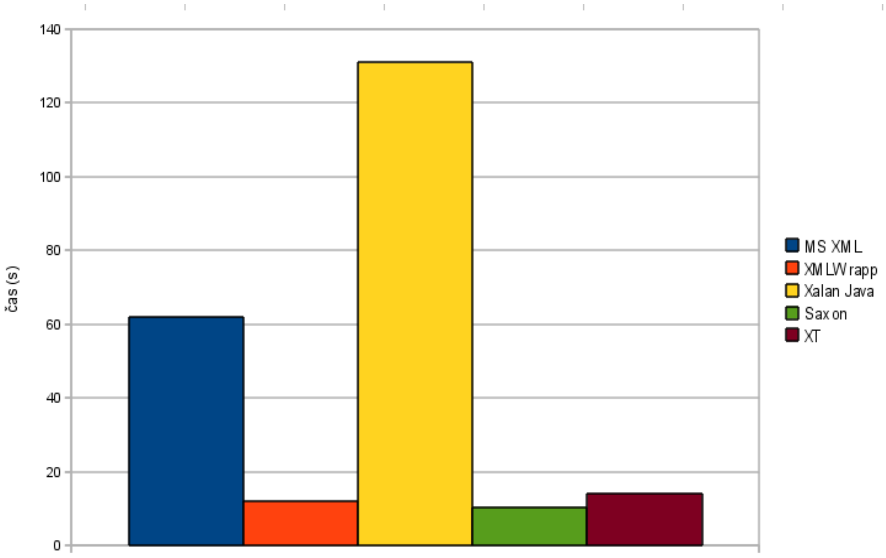
Pomerne prekvapivé boli vysoké časy, ktoré dosiahli MSXML (61,98 sekúnd) a hlavne Xalan-Java, ktorý s časom 130,93 sekúnd potreboval takmer 13-násobok času oproti najrýchlejšiemu procesoru.

V tomto teste sa prejavili obrovské rozdiely medzi jednotlivými procesormi, keď rozdiel medzi najrýchlejším a najpomalším bol 120 sekúnd.

Výsledky testovania malých súborov sú v tabuľke 4 a na obrázku 7. Ako je možné vidieť v tabuľke, najrýchlejšie zvládol spracovať súbory procesor Xt, ktorému to trvalo

Súbor	MSXML	XMLWrapp	Xalan J	Saxon	XT
SwissProt	61,98 s	11,98 s	130,93 s	10,42 s	14,14 s

Tabuľka 3: Súbor SwissProt.xml — Výsledky



Obr. 6: Graf výsledných časov – velký súbor

7,46 s. Procesory Xalan a Saxon dosiahli takmer zhodný čas okolo 9,5 s. Procesor MSXML potreboval na spracovanie 10,12 sekúnd a XML Wrapp 11,07 sekúnd.

Test ukázal, že pri spracovávaní malých súborov sú procesory podstatne vyrovnanejšie ako pri testovaní veľkého súboru. Rozdiel medzi najrýchlejším a najpomalším procesorom bol približne 50 %.

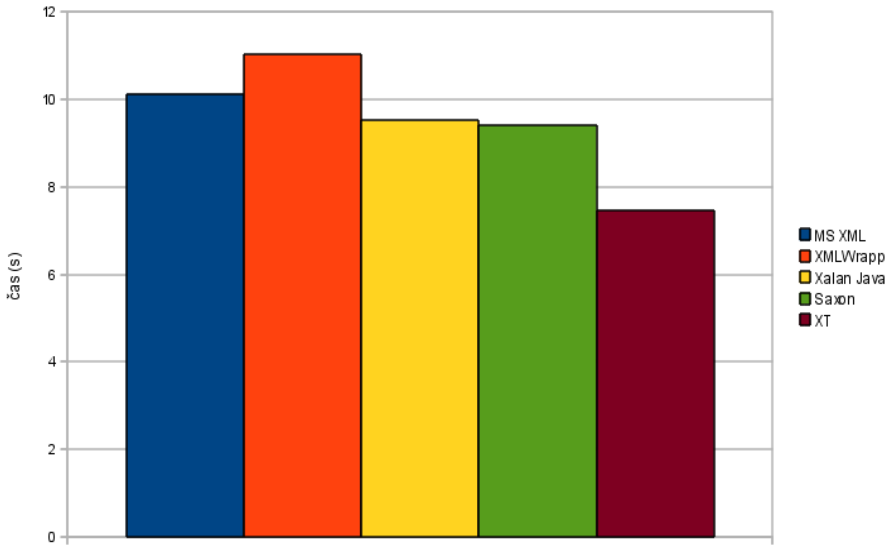
Zaujímavosťou je, že procesory pre jazyk C++ sa umiestnili až na posledných dvoch miestach, čo potvrzduje, že oveľa viac ako rýchlosť jazyka zohráva v tomto prípade úlohu efektívnosť implementácie tej-ktorej úlohy, ktorá sa prejavuje v špecifických prípadoch.

Súbor	MSXML	XMLWrapp	Xalan J	Saxon	XT
Wiki	10,12 s	11,07 s	9,51 s	9,4 s	7,46 s

Tabuľka 4: Malé súbory — Výsledky

4.3 Podmienené spracovanie

Výsledky podmieneného spracovávania veľkého súboru sú v tabuľke 5 a prehľadne na obrázku 8. Z procesorov pre jazyk Java bol najrýchlejší Saxon s časom 9,61 sekúnd. O niečo pomalší bol Xt s časom 12,89 sekúnd. Výrazne najpomalší bol Xalan-Java ktorému spraco-



Obr. 7: Graf výsledných časov – malé súbory

vanie tohto súboru trvalo vyše 143 sekúnd. Paradoxne sa spracovanie súboru procesormi Saxon a Xt zrýchlilo, ale Xalan-Java bol ešte pomalší.

Procesor MSXML znova nepríjemne prekvapil. S časom 54,37 sekúnd je síce rýchlejší ako pri nepodmienenom spracovávaní, ale tento čas je stále príliš vysoký. XMLWrapp si o 25 % pohoršil a skončil s časom 15,81 sekúnd.

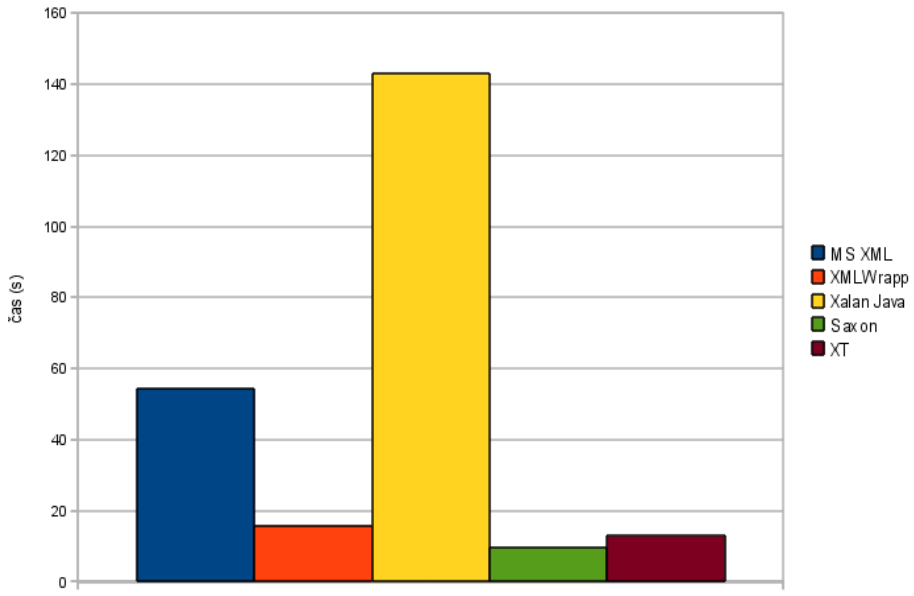
Tento test opäť preukázal problémy procesorov MSXML a Xalan-Java pri spracovávaní veľkého súboru. MSXML trvalo spracovanie vyše paťnásobok času, ktorý potreboval najrýchlejší procesor. Xalan potreboval dokonca vyše 14-krát toľko času. Dokonca potreboval viac času ako pri nepodmienenom spracovávaní, hoci elementov, ktoré bolo treba spracovať, bolo menej.

Súbor	MSXML	XMLWrapp	Xalan J	Saxon	Xt
SwissProt	54,37 s	15,81 s	143,02 s	9,61 s	12,89 s

Tabuľka 5: Súbor SwissProt.xml — Výsledky

Výsledky podmieneného spracovania malých súborov sú v tabuľke 6 a v grafickej podobe na obrázku 9. Najlepšie obstál procesor Saxon, ktorému to trvalo takmer rovnaký čas, ako nepodmienené spracovanie. Procesor Xalan-Java si mierne pohoršil a je z procesorov pre jazyk Java najpomalší. O takmer 25 % sa zhoršil procesor Xt a dostal sa až za procesor Saxon.

Procesory pre jazyk C++ sa umiestnili až na posledných dvoch miestach. MSXML bol výkonnejší a na spracovanie potreboval 10,97 sekúnd. Na poslednom mieste sa umiestnil XMLWrapp s časom 13,24 s.

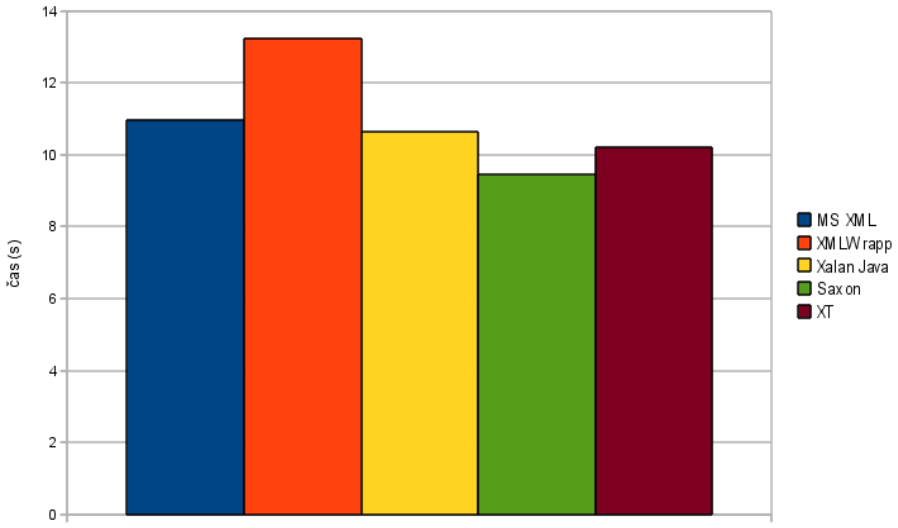


Obr. 8: Graf výsledných časov – podmienené spracovanie veľkého súboru

Rozdiel medzi prvým a posledným procesorom je približne 35 % a potvrdzuje sa že pri spracovávaní malých súborov nie sú rozdiely veľmi veľké. Na rozdiel od veľkého súboru však všetky procesory mali o niečo horší čas ako pri nepodmienenom spracovávaní. Väčšinou to nebol výrazný rozdiel, iba v prípade procesora Xt tento rozdiel bol 25 %.

Súbor	MSXML	XMLWrapp	Xalan J	Saxon	Xt
Wiki	10,97 s	13,24 s	10,64 s	9,44 s	10,21 s

Tabuľka 6: Malé súbory — Výsledky



Obr. 9: Graf výsledných časov – podmienené spracovanie malých súborov

5 Záver

Cieľom tejto práce bolo otestovanie rôznych implementácií XSLT procesorov na rôznych vstupných dátach a transformačných súboroch. Hlavným prínosom tejto práce je, že procesory boli otestované na reálnych XML súboroch, ktoré sa používajú v praxi a neboli špeciálne vytvorené pre účely tohto testovania, testovali sa súbory rôznych veľkostí a niekoľko transformačných šablón.

Z procesorov pre jazyk Java na testovaní veľkého súboru predviedol veľmi zlý výkon procesor Xalan-Java, ktorému trvalo spracovanie súboru 10–13-krát dlhšie, ako zvyšným dvom procesorom. Paradoxnou situáciou bolo, že procesor na rozdiel od ostatných potreboval viac času na podmienené spracovanie dokumentu, kde sa niektoré elementy nespracovávali vôbec. Takéto veľmi pomalé spracovanie sa neskôr potvrdilo aj pri použití samostatnej aplikácie (jar archívu), ktorú dodávajú sami vývojári. Pri testovaní malých súborov sa však tento procesor rýchlosťou vyrovnal ostatným.

Podobne pri testovaní veľkého súboru neuspel procesor z balíka MSXML. Takisto potreboval podstatne väčší čas ako iné procesory. Pri podmienenom spracovávaní už potreboval menej času, ale stále to bolo veľa.

Z nameraných výsledkov vyplýva, že na spracovanie veľkých súborov (veľkosť okolo 100 MB) sú z testovaných procesorov vhodné najmä Saxon a Xt. Pri malých súboroch sa situácia vyrovnáva a v rýchlosti jednotlivých procesorov nie sú žiadne priepastné rozdiely.

V rámci tejto práce neboli testované veľmi veľké súbory (rádovo niekoľko stoviek MB a viac) z dôvodu pamäťovej náročnosti. Táto pamäťová náročnosť je daná princípom fungovania XSLT procesorov. XSLT procesor si pred spracovaním dokumentu z celého dokumentu zostaví strom, ktorý si udržiava v pamäti. Tento strom je väčší ako pôvodný XML dokument. Ďalšia pamäť je potrebná na samotné vykonanie transformácie (keďže XSLT procesor nemodifikuje priamo vstupný strom). Preto spotreba pamäte rastie ako približne lineárna funkcia veľkosti súboru.

Možným riešením je použiť nejaký procesor, ktorý zvláda prúdové spracovanie dokumentu (napr. Saxon EE [14]) alebo jazyk, ktorý nepotrebuje vytvárať strom v pamäti. Takýmto jazykom sa snaží byť jazyk STX, založený na XQuery 1.0 a XPath 2.0, ktorý sa však momentálne nachádza iba vo fázi návrhu. [30]

6 Literatúra

[1] *XSL History* [online], c1999-2007 [cit. 2010-04-03]. Dostupné z WWW: <http://www.dpawson.co.uk/xsl/sect1/history.html>

[2] *XSL Transformations (XSLT)* [online], 16 November 1999 [cit. 2010-01-20]. Dostupné z WWW: <http://www.w3.org/TR/xslt>

[3] *XSL Transformations (XSLT) Version 1.1* [online], 24 August 2001 [cit. 2010-01-20]. Dostupné z WWW: <http://www.w3.org/TR/xslt11>

[4] *XSL Transformations (XSLT) Version 2.0* [online], 23 January 2007 [cit. 2010-01-20]. Dostupné z WWW: <http://www.w3.org/TR/xslt20>

[5] FITZGERALD, Michael. *Learning XSLT*. Sebastopol: O'Reilly, 2003. 368 s. ISBN 0-596-00327-7.

[6] *XSLT Browsers* [online], c1999-2010, [cit. 2010-04-27]. Dostupné z WWW: <http://www.w3schools.com/XSL/xsl_browsers.asp>

[7] *XML Path Language (XPath)* [online], 16 November 1999, [cit. 2010-04-03]. Dostupné z WWW: <http://www.w3.org/TR/xpath>

[8] *well formed - XML Tutorial* [online], c1998-2004, [cit. 2010-04-03]. Dostupné z WWW: <http://www.javacommerce.com/displaypage.jsp?name=wellform.sql&id=18238>

[9] MARCHAL, Benoit. *How an XSLT processor works* [online], 15 Mar 2004, [cit. 2010-03-23]. Dostupné z WWW: <http://www.ibm.com/developerworks/xml/library/x-xslang/>

[10] *alphaWorks : LotusXSL : Overview* [online], January 8, 1999 [cit. 2010-04-27]. Dostupné z WWW: <http://www.alphaworks.ibm.com/formula/lotusxsl/>

[11] *Xalan-Java Overview* [online], c2006 [cit. 2010-03-23]. Dostupné z WWW: <http://xml.apache.org/xalan-j/overview.html>

[12] *Xalan-C++ Overview* [online], c1999-2004 [cit. 2010-03-23]. Dostupné z WWW: <http://xml.apache.org/xalan-c/overview.html>

[13] *Saxon XSLT - Wikipedia, the free encyclopedia* [online], 11 March 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=Saxon_XSLT&oldid=349304156>

[14] KAY, Michael H. *The SAXON XSLT and XQuery Processor* [online], 2010-01-27 [cit. 2010-03-23]. Dostupné z WWW: <http://saxon.sourceforge.net/>

[15] *XT* [online], c2002–2005 [cit. 2010-03-23]. Dostupné z WWW: <http://www.blnz.com/xt/xt-20051206/index.html>

[16] *MSXML* [online], c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms763742.aspx>

[17] *MSXML - Wikipedia, the free encyclopedia* [online], 24 April 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=MSXML&oldid=357998802>

[18] FLECK, John. *libxslt Tutorial* [online], c2001, [cit. 2010-04-03]. Dostupné z WWW: <http://xmlsoft.org/XSLT/tutorial/libxslttutorial.html>

[19] *The WebKit Open Source Project* [online], dátum neznámy [cit. 2010-05-04]. Dostupné z WWW: <http://webkit.org/projects/xslt/index.html>

[20] *Unicorn Enterprises SA* [online], c2000-2003 [cit. 2010-05-05]. Dostupné z WWW: <http://www.unicorn-enterprises.com/products.html>

[21] *What is C++?* [online], 23 January 2009 [cit. 2010-05-05]. Dostupné z WWW: <http://www.hitmill.com/programming/cpp/whatiscpp.html>

[22] *Java Tips - What is Java Just in Time Compiler* [online], c2005 - 2008 [cit. 2010-05-05]. Dostupné z WWW: <http://www.java-tips.org/java-se-tips/java.lang/what-is-java-just-in-time-compiler.html>

[23] *Stefan Krause.blog() » Blog Archive » Update For Java Benchmark* [online], July 2nd, 2008, [cit. 2010-05-05] Dostupné z WWW: <http://www.stefankrause.net/wp/?p=9>

[24] *jaxp: JAXP Reference Implementation* [online], c2010, [cit. 2010-03-23] Dostupné z WWW: <https://jaxp.dev.java.net/>

[25] MCLAUGHLIN, Brett D. *All about JAXP, Part 2* [online], 17 May 2005, [cit. 2010-03-23]. Dostupné z WWW: <http://www.ibm.com/developerworks/xml/library/x-jaxp2/>

[26] *Introduction to JAXP 1.1 (Java and XSLT)* [online], c2002, [cit. 2010-03-23]. Dostupné z WWW: <http://docstore.mik.ua/orelly/xml/jxslt/ch05_02.htm>

[27] Káldy, Robert. *Component Object Model* [online], c2003, [cit. 2010-04-27]. Dostupné z WWW: <http://antonio.cz/com/2.html>

[28] *ExPASy - UniProt Knowledgebase: Swiss-Prot and TrEMBL* [online], Last modified 20/Apr/2010, [cit. 2010-04-27]. Dostupné z WWW: <http://www.expasy.org/sprot/>

[29] *Wikipedia XML Corpus* [online], [cit. 2010-04-27]. Dostupné z WWW: <http://www-connex.lip6.fr/ denoyer/wikipediaXML/>

[30] *Streaming Transformations for XML (STX) Version 1.0* [online], [cit. 2010-05-05]. Dostupné z WWW: <http://stx.sourceforge.net/documents/spec-stx-20070427.html/>

A Obsah pripojeného CD

Adresár	Obsah
/text/	Práca vo formáte PDF a zdrojové kódy systému LaTeX
/source/	Zdrojové kódy testovacieho rozhrania
/bin/	Skompilované testovacie rozhranie
/xml/	Použité XML súbory a transformačné súbory XSLT
/processors/	Použité XSLT procesory